(12) UK Patent Application (19) GB (11) 2 332 542 (13) A

(43) Date of A Publication 23.06.1999

(21) Application No 9726886.6

(22) Date of Filing 20.12.1997

(51) INT CL[6]
G06F 7/52 7/72

(52) UK CL (Edition Q)
G4A ACX AMD A2BX A2B2 A2B3 A2B4
U1S S2120

(56) Documents Cited
EP 0566438 A2

(58) Field of Search
UK CL (Edition P) G4A ACX AMD
INT CL[6] G06F

(71) Applicant(s)
Motorola Limited
(Incorporated in the United Kingdom)
Jays Close, Viables Industrial Estate, BASINGSTOKE, Hampshire, RG22 4PD, United Kingdom

(72) Inventor(s)
Peter McGinn
Russell Domenic Hobson

(74) Agent and/or Address for Service
Sarah Gibson
Motorola Limited, European Intellectual Property Operation, Midpoint, Alencon Link, BASINGSTOKE, Hampshire, RG21 7PL, United Kingdom

(54) Abstract Title
Multiplication circuitry

(57) A data processing system for performing multiplication, e.g. modular multiplication as used in implementing the Montgomery Reduction Algorithm, includes a multiplier 20 which simultaneously performs two multiplications and sums the results of the two multiplications. For example, multiplier 20 performs A.B + Y.N simultaneously by supplying B and N values to the multiplier from RAM portions 12 and 14, and selectively accumulating either A, Y, A+Y, or 0 as stored in register 22 depending upon the bit values of B and N in each row of the multiplier. In performing a 512 RSA encryption, multiplier 20 is preferably a 16x17 parallel multiplier.

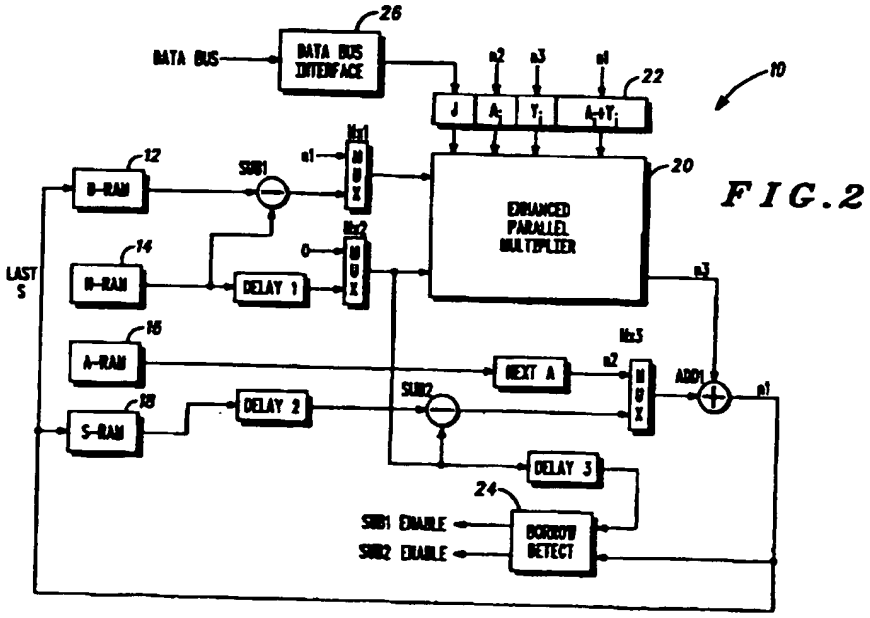


FIG.2

GB 2 332 542 A

DATA PROCESSING SYSTEM FOR PERFORMING MULTIPLICATION AND MULTIPLICATION METHOD

FIELD OF THE INVENTION

This invention relates generally to a data processing system for performing multiplication using a multiplier, and more particularly for performing modular multiplication such as used in implementing the Montgomery Reduction Algorithm.

BACKGROUND OF THE INVENTION

Modular multiplication is extensively used in implementing cryptographic methods such as RSA cryptography.

The Montgomery algorithm is one of the most efficient techniques for performing modular multiplication. Its use is particularly effective where high performance is required so as to minimise the computation time.

The Montgomery proof is given in Appendix 1 and the Montgomery Reduction Algorithm is outlined below:

Montgomery Algorithm

To enact the *P* operator on A.B we follow the process outlined below:

(1) $X = A.B + S$ (S initially zero)

(2) $Y = (X.J) \bmod 2^n$ (where J is a pre-calculated constant, and n is the number of bits in N)

(3) $Z = X + Y.N$

(4) $S = Z/2^n$

(5) $S = S \pmod{N}$ (N is subtracted from S, if $S \geq N$; else $S = S$)

Thus $S = P = P(A.B)_N$ (the result in the Montgomery Field of numbers)

In financial applications where smartcards are used as a means of ensuring a high level of security during the transaction, Public key cryptography is becoming increasingly popular. Public key cryptography offers a higher level of protection than the traditional symmetric or private key methods, but until recently has been expensive to implement. Advances in technology have now made the implementation of such methods cost effective. RSA Public Key capability has been designed into smartcard microcontrollers which also include an on-chip co-processor which has been specifically designed to perform modular multiplications for operands each of 512 bit length. The co-processor is directly driven by the microcontroller's CPU under software control by a program stored either in ROM or in EEPROM. Such a co-processor which implements the Montgomery algorithm for modular reduction without the division process is known from European Patent Publication EP-0601907-A.

In performing the multiplications of equations (1) and (3) above, two separate 32 x 1 bit multipliers (ML1 & ML2 of FIG. 1) are used. Thus, to perform a 512 x 512 bit multiplication, 16 rotations are necessary, with an approximate consumption of 544 clock cycles for each rotation. Furthermore, the operands are derived from values B, S, and N received from fixed length shift registers. Accordingly, such an architecture will be unable to efficiently perform multiplications on operands of greater than 512 bits. The silicon area required to expand the architecture to accept larger operands is prohibitive. Furthermore, there is an increasing demand for faster cryptographic operations

such that 8000+ clock cycles to perform a 512 x 512 multiplication is unacceptable.

Therefore, a need exists for an improved data processing system which is capable for performing multiplication in fewer clock cycles than in prior art systems. Further, it is desirable that such a system be readily adaptable to operands of greater length while requiring minimal silicon area to implement.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a block schematic diagram of a known, prior art co-processor for performing modular multiplication to implement the Montgomery Reduction Algorithm;

FIG. 2 shows a block schematic diagram of an improved data processing system for performing multiplication, e.g. modular multiplication as is used to implement the Montgomery Reduction Algorithm;

FIG. 3 shows a simplified block schematic diagram demonstrating the inputs and outputs of the multiplier used in the data processing system of FIG. 2;

FIG. 4 is a chart demonstrating the values added by the multiplier used in the data processing system of FIG. 2, when used to perform modular arithmetic for implementing the Montgomery Reduction Algorithm using four operands.

FIG. 5 shows a block schematic diagram of a portion of the multiplier used in the data processing system of FIG. 2 which demonstrates one

embodiment for selecting which values are to be added in performing the multiplication of FIG. 4.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Known Co-processor Operation

FIG. 1 shows a diagram of a known, prior art hardware implementation of a data processing system, in the form of a co-processor, which performs the Montgomery algorithm for both full mode 512 bit and half-mode 256 bit operands.

The diagram shows the execution unit which comprises basically three 512 bit clocked shift registers (Shift Registers B, S, and N) and two parallel - serial multipliers (ML1 and ML2).

The B value and the modulus N are preloaded into the B and N registers respectively. Register S is used to store the intermediate result after each rotation of approximately 544 clock cycles (512 to perform the multiplication and another 32 to shift the last 32 bit value out of the register). Initially the S register will be cleared. The pre-calculated Montgomery Constant, J_0, is loaded into the co-processor via a 32 bit shift register and latched in Latch2.

The A value is shifted in 4 bytes (32 bits) at a time, (A_i) via multiplexer M2_1;2 and latched in Latch1. The value in the B register is serially clocked one bit at a time into a first parallel - serial multiplier ML1. The output of this multiplier, at node n_A, is the value $A_i.B$. The value $A_i.B$ is then summed at adder AD1 to the intermediate value stored in register S to produce the value $X = A_i.B + S$, at node n_B.

For the first 32 clock cycles, the first 32 bit portion of the X value is fed via multiplexer M3_1;4 into a second parallel - serial multiplier ML2, where it is multiplied by the value J_0. The output from ML2 at node n_D is the value $Y_0 = X.J_0$. Y_0 is fed back through a 32 bit shift register and latched in Latch2 via multiplexer M.

After the first 32 clock cycles, multiplexer M3_1;4 switches and feeds the modulus N into the multiplier ML2, where N is multiplied by Y_0 to produce the value $Y_0.N$. This value is then summed, over the next 544 clock cycles, with X at adder AD2 to produce the value $Z = X + Y_0.N$. The least significant 32 bits of this calculation are zero and only the 512 most significant bits are saved back in the S register. This completes one full rotation.

Sixteen rotations, using a 32 bit multiplication, are required to perform the full 512 bit by 512 bit multiplication, which gives:

$$P = A.B.I \pmod{N} = P(A.B)_N$$ (the result in the Montgomery Field of numbers).

To recover the required result, P is multiplied by H (a pre-calculated Montgomery constant) to give the result in the field of real numbers:

$$R = A.B \pmod{N} = P(P.H)_N$$

Various improvements have been made to the co-processor architecture shown in FIG. 1. For example, instead of using a single serial loop clocking stream, the architecture can be modified to use bit-pair multiplication, addition, and subtraction, where the adders, subtractors, and parallel-serial multipliers compute results two bits at a time. This improvement immediately doubles the performance for the same clock

frequency. For further explanation of this improvement, refer to GB Application No. 9622719.4.

A further improvement is achieved by replacing the 512 bit clocked, serial shift registers with a combination of random access memory (RAM), a parallel to serial interface, and an 8 bit clocked shift register. The advantages of such a replacement include 1) reduced power consumption, 2) greater flexibility in handling varying operand lengths, and 3) the ability to perform exponentiation without intervention by the CPU. Further explanation of the RAM-based improvement can be found in GB Application No. 9622714.5.

Yet another improvement to the architecture shown in FIG. 1 is the use of a single 64 bit clocked serial multiplier to calculate A.B and N.Y on alternate clock phases but still within a single clock cycle. Use of such a single multiplier increases performance (only half the number of rotations are needed) without increasing the clock frequency, while consuming only slightly more silicon area than using two 32 x 2 bit multipliers. Further description of the implementation of this single multiplier implementation can be found in GB Application No. 9701958.2.

Despite the various improvements which have been made or proposed to the co-processor architecture of FIG. 1, there continues to be a need for more efficient multiplication.

Imoroved Data Processing System

Referring now to FIG. 2, an improved data processing system 10 for performing multiplication is shown. In a preferred embodiment, data processing system 10 is also in the form of an integrated circuit co-processor suitable for inclusion in a smartcard. System 10 includes random

access memory (RAM) portions 12, 14, 16, and 18 for storing values of B, N, A, and S, respectively, as shown. For purposes of describing the invention, the assumption is made that B, N, A, and S values are 512 bits, however the invention is not limited to this particular size. In such an embodiment, RAM portions 12-18 are preferably 512 bit arrays, but could be 1024 bits arrays or larger.

System 10 further includes a multiplier 20. In a preferred embodiment, multiplier 20 is a parallel multiplier. If values A, B, and N are 512 bits, multiplier 20 is preferably a 16 row by 17 column (hereinafter, 16 x 17) multiplier which performs 16 bit by 16 bit multiplication. In performing a multiply operation of 16 bit x 16 bit, multiplier 20 should accommodate a 33 bit wide result (the extra bit is needed to accommodate the accumulation of A+Y as explained further below). Alternatively, multiplier 20 could be a 32 x 33 bit multiplier for multiplying 32 bits of each value at a time, or even larger, with correspondingly larger accumulator/output stage. However, for the remainder of the description it is assumed that data processing system is used to perform 16 bit multiplication.

Multiplier 20 is used to simultaneously multiply A.B and Y.N and to sum the results in accordance with the present invention. To achieve this, as explained further below, values J, A, Y and A+Y are input into the multiplier via a register 22. The B value is fed to the multiplier through a first subtractor, Sub1, and a first multiplexer, Mx1, the purposes of which are explained further below. The N value is fed to the multiplier through a second multiplexer, Mx2. Mx1 and Mx2 are sized according to the size of the multiplier.

Data processing system 10 also includes a second subtractor, Sub2, for subtracting the value N from S (see equation (5)). An adder, Add1, is included to calculate Z

as the sum of the output of multiplier 20 (A.B + Y.N) and S (see equation (3)). Add1 is also used to calculate A+Y as explained further below. Borrow detection circuitry 24 is used to determine whether N is to be subtracted from S, and thus is used to enable or disable Sub1 and Sub 2 as indicated.

In performing the Montgomery Reduction Algorithm, data processing system 10 works as follows. J, having a bit width which matches the width of multiplier 20 (in this case a 16 bit value), is precalculated Montgomery Constant which is loaded into register 22 from the central processing unit (CPU) of the system via a data bus interface 26. The J value remains constant throughout each 512 x 512 multiplication and is held in a 16 bit register field. Value A is latched into register 22 16 bits at a time from node n2 (thus A is denoted in the figures as A_i), which receives the value 16 bits at a time directly from RAM portion 16.

Value Y must be calculated as in equation (2) for each new A valued loaded into register 22 (thus Y is denoted in the figures as Y_i). Accordingly, X must be calculated to be able to determine the product X.J. From equation (1), X = A.B + S. Only 16 bits of Y are needed, therefore 16 bits of A and 16 bits of B are multiplied to achieve Y, with Add1 being fed the 16 least significant bits (LSBs) of the A.B result. By setting the Mx2 output to be the "0" input, multiplier 20 calculates A.B. Value A is received as described above. Value B is received from RAM portion 12, with or without subtraction by Sub1 as dictated by borrow detection circuitry 24. The output of the multiplier at n3 is thus A.B. The 16 LSBs of the result are fed to Add1. S is added to this product at Add1 to produce A.B + S at node n1. This node is fed to Mx1 such that on the next clock cycle, n1 rather than a B value is fed into multiplier 20. Multiplier 20 then calculates the value at n1 (A.B. + S) times J (from

register 22). The 16 LSBs of the product X.J are then fed to register 22 as the Y_i value of FIG. 2.

The value A+Y is calculated in the same clock cycle that Y_i is fed to register 22 by using Add1 to add A_i to the Y_i value at n3 to produce A_i+Y_i at node n1. The n1 value is then fed to register 22. Note that the register values for each of J, A, and Y are 16 bits wide, while that of A+Y is 17 bits wide because the addition of two 16 bit values may produce a 17 bit value.

With the implementation illustrated and described, it will take approximately 7 clock cycles to load J, A, Y, A+Y into register 22 before multiplication of A.B + Y.N can begin. Once the values in register 22 are set, data processing system 10 operates to determine Z. From equations (1) and (3), Z = S + A.B + Y.N. In accordance with the present invention, A.B + Y.N is calculated simultaneously using a single parallel multiplier.

As represented by FIG. 3, the N value and B value are input into multiplier 20 as the two multiplicands of A.B + Y.N. A and Y values are input as the multipliers of A.B + Y.N. In accordance with the present invention, the additional input of A+Y is input into multiplier 20 to perform the operation A.B + Y.N. In its simplest form, the operation can be broken down into a series of additions and shifts. When a bit of B is "0", nothing is added to the accumulated result of the multiplier. When a bit of B is "1", A is added to the accumulated result of the multiplier. Similarly, when a bit of N is "0", nothing is added to the accumulated result of the multiplier. When a bit of N is "1", Y is added to the accumulated result of the multiplier. When both corresponding bits of B and N are "1", both A and Y must be added.

In accordance with one embodiment of the invention, B and N are fed to multiplier 20 16 bits at a time and are multiplied with 16 bits of A and Y as described above. Following this multiplication, the next 16 bits of B and N are fed to multiplier 20 and are multiplied with the same 16 bits of A and Y already stored in register 22. The process continues until all 512 bits of B and N have been multiplied by 16 bits of A and Y. This completes one rotation of the multiplication. In total the rotation takes approximately 43 clock cycles: about 7 to load register 22; 32 to compute $A_i.B + Y_i.N$; and about 4 to get the data through the various adders and substractors.

After the completion of one rotation, the next 16 bits of A are fed to register 22 and Y and A+Y are recalculated using the new A value as described above. J is unchanged. After the new register values are set, B and N are again fed to multiplier 20 16 bits at a time, until all 512 bits of B and N have been multiplied using the new register values. This completes a second rotation. Rotations are repeated until all 512 bits of A have been used, resulting in a total of 32 rotations for a 512 x 512 multiplication, which corresponds to about 1376 clock cycles. This performance is a 7x improvement over the prior implementation described in reference to FIG. 1 and a 3.5x improvement over the use of two 2x32 multipliers. If the present invention is implemented using a 32x33 parallel multiplier, only 432 clock cycles are needed to perform a 512 x 512 multiplication because only 16 rotations of approximately 27 clock cycles are required.

Throughout a 512 x 512 multiplication, an intermediate S value is generated. Initially, S=0. After the first 16 x 16 multiplication of A.B + Y.N, the 16 LSBs of the intermediate result are fed to Add1 and summed with the last S value. Mx3 controls what is added at Add1. As described above, Mx3 initially passes the next A value to Add1 to produce the A+Y value stored in register 22.

During the multiplication of A.B and Y.N, Mx3 then passes the previous S value from S-RAM portion 18 to Add1. The result from Add1 is fed to S-RAM portion 18 as a new S value. Each time S is added at Add1, borrow detection circuitry is used to determine if the new value of S is greater than N. If greater, borrow detection circuitry 24 enables Sub2 for the next rotation, and N is subtracted from S before the value is added by Add1. After the entire 512 x 512 multiplication has been performed, the last S value is stored in B-RAM portion 12 because S is sometimes used in subsequent calculations as the B value. Again, however, N must be subtracted from this value if S>N. Thus, Sub1 is included. Sub1 is enabled by borrow detection circuitry 24 during such subsequent calculations if the S value stored in B-RAM portion 12 is greater than the N for the former calculation.

Also throughout the multiplication, the A, Y, A+Y and 0 values are accumulated in the multiplier. In accordance with the present invention, the 17 most significant bits (MSBs) of each 33 bit multiplication result are carried in the accumulator of the multiplier, while the 16 LSBs are fed to Add1 as explained above.

An example of the operation of multiplier 20 is shown below in FIG. 4. An arbitrary B value of 00111100 is supplied to the multiplier. An arbitrary N value of 10101010 is supplied to the multiplier. To perform A.B + Y.N, the bits of B and N are examined to determine what value is to be added to the accumulator result stage of the multiplier. In this example, the LSBs of B and N are "0" so nothing is added, as indicated in FIG. 4. The next LSBs of B and N are "0" and "1", respectively, so Y is added. The next are "1" and "0", respectively, so A is added. The next are "1" and "1" so A and Y are both added using the precalculated A+Y value. These examples demonstrate the four possible combinations of B and N bit comparisons. The addition of either 0, A, Y, or A+Y is

determined for each B-N bit pair, but is done in a single clock cycle for whatever B & N size value is supplied to the multiplier.

A more particular example is shown below in reference to TABLE 1. For simplicity, only 4 bit arbitrary values of B, N, A, Y will be used as follows. The number in parentheses is the decimal equivalent of the binary value.

B = 1001	(9)
N = 1100	(12)
A = 1010	(10)
Y = 1101	(13)
A+Y = 10111	(23)

N	B	Add								
0	1	A					1	0	1	0
0	0	0				0	0	0	0	
1	0	Y			1	1	0	1		
1	1	A+Y	1	0	1	1	1			
			1	1	1	1	0	1	1	0

TABLE 1

Thus, A.B + Y.N equals 11110110, or 10.9 + 13.12 = 246.

In order to implement the operation of multiplier 20 as described above, circuitry is needed to control which of A, Y, A+Y, or 0 is added. One embodiment for such control is illustrated in FIG. 5. FIG. 5 is a portion of the multiplier showing decode circuitry 30 associated with each row of the multiplier. In one form, the decode circuitry may comprise a 3:1 multiplexer 32 for each row and a switch 34 for each bit. Multiplexers 32 are used to control which of A, Y, or A+Y is active for each row (i.e. which value is passed through each switch in the respective row). The output of the multiplexer is

determined by the inputs of B and N bit values as described above. If all of A, Y, and A+Y are inactive (i.e. both the B and the N bit values are 0), 0 is passed.

It is noted that the circuitry described and shown in reference to FIG. 5 is but one example of how a multiplier could be implemented in accordance with the present invention and should not be viewed as limiting the scope of the invention.

As mentioned above, a 16x17 multiplier is preferred to perform a 512 bit encryption (assuming 16 bit calculations per clock). The 17th column of the multiplier is really "overhead" that is needed because A+Y can produce a 17 bit value. The area consumed by such a multiplier is estimated to be about 2924 gate equivalents, calculated as follows. An NxM multiplier would include (N-1)(M) full adders, each with a gate equivalent of 8; M half adders, each with a gate equivalent of 4; and NxM 3:1 multiplexers, each with a gate equivalent of 3. In contrast, the implementation of two 2x32 serial multipliers as done in the prior art is approximately 2700 gate equivalents. Thus, for a slight increase in silicon area, the performance of the multiplier can be significantly improved. Two 2x32 serial multipliers can perform the 512 x 512 bit multiplications in approximately 4864 clocks (512/2 bits multiplied per clock + 48 overhead clock cycles, for each of 16 rotations), whereas a 16x17 parallel multiplier in accordance with the present invention requires only approximately 1376 clocks, as explained above. If silicon area is available to use a 32x33 parallel multiplier in accordance with the present invention, the calculation can be done in just 432 clocks because only half as many rotations are needed.

The foregoing description and illustrations contained herein demonstrate many of the advantages associated with

the present invention. In particular, it has been revealed that a multiplier can be used to perform the entire operation of A.B + Y.N, as is needed in performing the Montgomery Reduction Algorithm. The use of a single parallel multiplier eliminates the need to increase clock speed to improve performance. Thus, there is no increase in power consumption by the data processing system, and design difficulties experienced at higher clock frequencies are avoided. Moreover, the present invention can be implemented in a silicon area comparable to that used in prior art processors which used two multipliers to perform the same operation, with improved performance. Furthermore, the present invention has significant silicon area savings as compared to using two 16x16 parallel processors, one to perform A.B and the other to perform Y.N. The only additional hardware needed to perform both operations with the same multiplier is an additional column in the multiplier (to accommodate the accumulation of A+Y) and some decode circuitry for each row to determine which of A, Y, A+Y or 0 is to be added. Moreover, the multiplier of the present invention can be used to perform traditional multiplications such as A.B, in addition to performing A.B + Y.N.

Thus it is apparent that there has been provided, in accordance with the invention, a data processing system having a multiplier and multiplication method using the same that fully meets the need and advantages set forth previously. Although the invention has been described and illustrated with reference to specific embodiments thereof, it is not intended that the invention be limited to these illustrative embodiments. Those skilled in the art will recognise that modifications and variations can be made without departing from the scope of the invention. For example, the present invention is not limited to any particular size of multiplication, nor is it required that the invention be used to implement the Montgomery Algorithm. While the values A, B, N, and Y are used in

the description and claims, such designations are for reference only. The present invention can be used to perform any calculation in the general form A.B + C.D. In addition, the invention is not limited to implementations having a parallel multiplier. A serial multiplier could be used, but would not have the performance of a parallel multiplier as herein described. Also, in the instance where Y is a function of the product of two other input values (e.g. A.B), it is not required that the same multiplier be used to calculate Y (to provide values Y and A+Y as multiplier inputs) as is used to perform A.B. + Y.N. A separate multiplier could be used to calculate Y, albeit with additional silicon consumption. Therefore, it is intended that this invention encompass all such variations and modifications as fall within the scope of the appended claims.

Appendix 1

Montgomery Modular Reduction Technique

The Montgomery function $P(A.B)_N$ performs a multiplication modulo N of the product A.B into the *P* field. The retrieval from the *P* field back into the normal modular field is performed by enacting P on the result of $P(A.B)_N$ and a precalculated constant H.

Thus if $P == P(A.B)_N$, then $P(P.H)_N == A.B \pmod{N}$.

Proof

We require to calculate $R = A.B \pmod{N}$.

First find Q, such that:

$$P2^n = A.B + Q.N \text{ (where N is odd)} \quad (1)$$

Note:

$$I.2^n == 1 \pmod{N} \text{ (and n is the bit length of N)} \quad (2)$$

Multiply equation (1) by I to give:

$$P.I.2^n = A.B.I + Q.I.N \quad (3)$$

Consider the left side of (3), from (2):

$$P.I.2^n == P \pmod{N} \quad (4)$$

Consider the right side of (3), then from (4):

$$P == (A.B.I + Q.I.N) \pmod{N}, \text{ and therefore:}$$
$$P == A.B.I \pmod{N} = P(A.B)_N \quad (5)$$

Consider $P(P.H)N$ then from (5):

$$P(P.H)_N == A.B.I^2.H \pmod{N} \quad (6)$$

Clearly if H is defined as I^{-2} then:

$$R == P(P.H)_N == A.B \pmod{N} \quad (7)$$

Equation (7) gives the desired result.

From (2) above, $H = 2^{2n} \pmod{N}$ and is a precalculated constant depending only on N and n.

It next requires that Q be found. From (1) it can be seen that:

$$\{A.B.I + Q.I.N\} \pmod{2^n} = 0 \qquad (8)$$

This implies:

$$A.B.I \pmod{2^n} = -Q.I.N \pmod{2^n}$$ and therefore,

$$Q == -N^{-1} A.B \pmod{2^n} \qquad (9)$$

For odd N, $J = N^{-1}$ such that $N.J = I.2^n + 1$.

Hence $Q == - A.B.J \pmod{2^n}$.

Note, J is also a precalculated constant depending only on N and n.

CLAIMS

1. A data processing system for performing multiplication comprising:

means for storing a plurality of binary data values, A, B, N, and Y; and

a multiplier having inputs for receiving values A, B, N, and Y from the respective means for storing said values, and further having an input for receiving a value of A+Y, wherein said multiplier computes an operation A.B + N.Y.

2. The data processing system of claim 1 wherein the multiplier is s parallel multiplier and is the only multiplier in the data processing system used to compute said operation.

3. The data processing system of claim 1 or 2 wherein the multiplier is used to compute Y, which is a function of a product of two of the other values.

4. The data processing system of claim 1, 2, or 3 wherein said values A, B, N, and Y are used to perform the Montgomery algorithm.

5. The data processing system of any preceding claim wherein said means for storing values B and N comprises random access memory, and wherein said means for storing values A and Y comprises a register means, said register means also used for storing the value A+Y.

6. The data processing system of any preceding claim further comprising decode circuitry coupled to the multiplier for selecting which values of A, Y, A+Y, or 0 are accumulated within the multiplier.

7. An integrated circuit which embodies the data processing system of any preceding claim.

8. A smartcard including an integrated circuit embodying the data processing system of any preceding claim.

9. A method for performing an operation A.B + Y.N comprising the steps of:

providing a data processing system having means for storing binary values A, B, Y, and N, a multiplier, and adder means;

using said adder means to calculate a binary value A+Y;

providing the values B and N as a first set of inputs to the multiplier, and providing values A, Y and A+Y as a switchable second set of inputs to the multiplier;

performing the operation A.B + Y.N using the multiplier by selectively accumulating either A, Y, A+Y, or 0 in the multiplier depending upon each bit value of B and N as input to the multiplier.

10. The method of claim 9 wherein the step of performing the operation A.B + Y.N is a part of performing the Montgomery Algorithm.

11. The method of claim 10 further comprising the step of using the multiplier to calculate Y as a function of a product of two of the other values.

12. An integrated circuit that performs the method of any preceding claim.

13. A smartcard including the integrated circuit of claim 11.

The Patent Office

21

Application No: GB 9726886.6 **Examiner:** Mike Davis
Claims searched: 1-13 **Date of search:** 1 July 1998

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.P): G4A (AMD, ACX)

Int Cl (Ed.6): G06F

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
X	EP 0566498 A2 (FORTRESS...)	1,9 at least

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

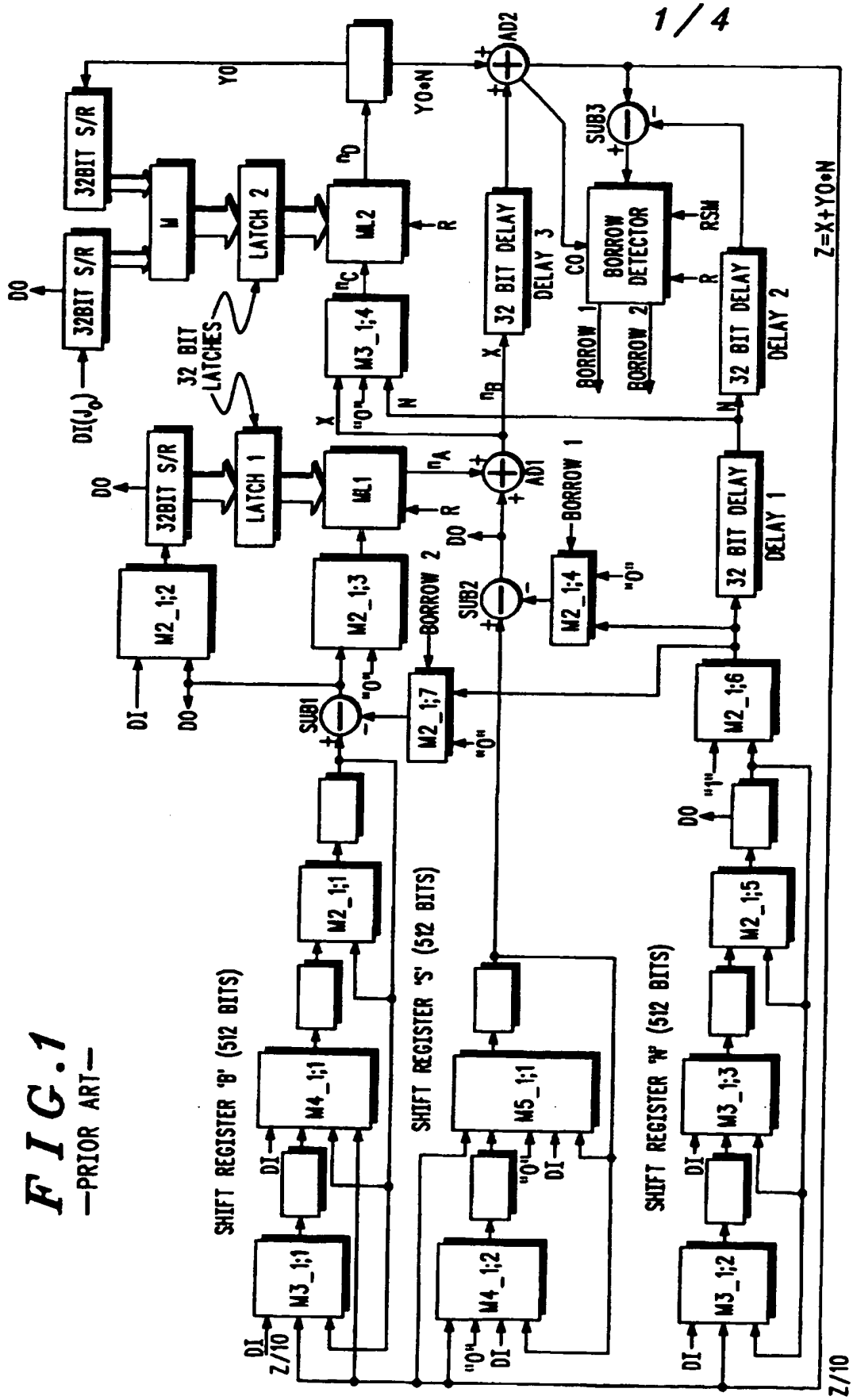
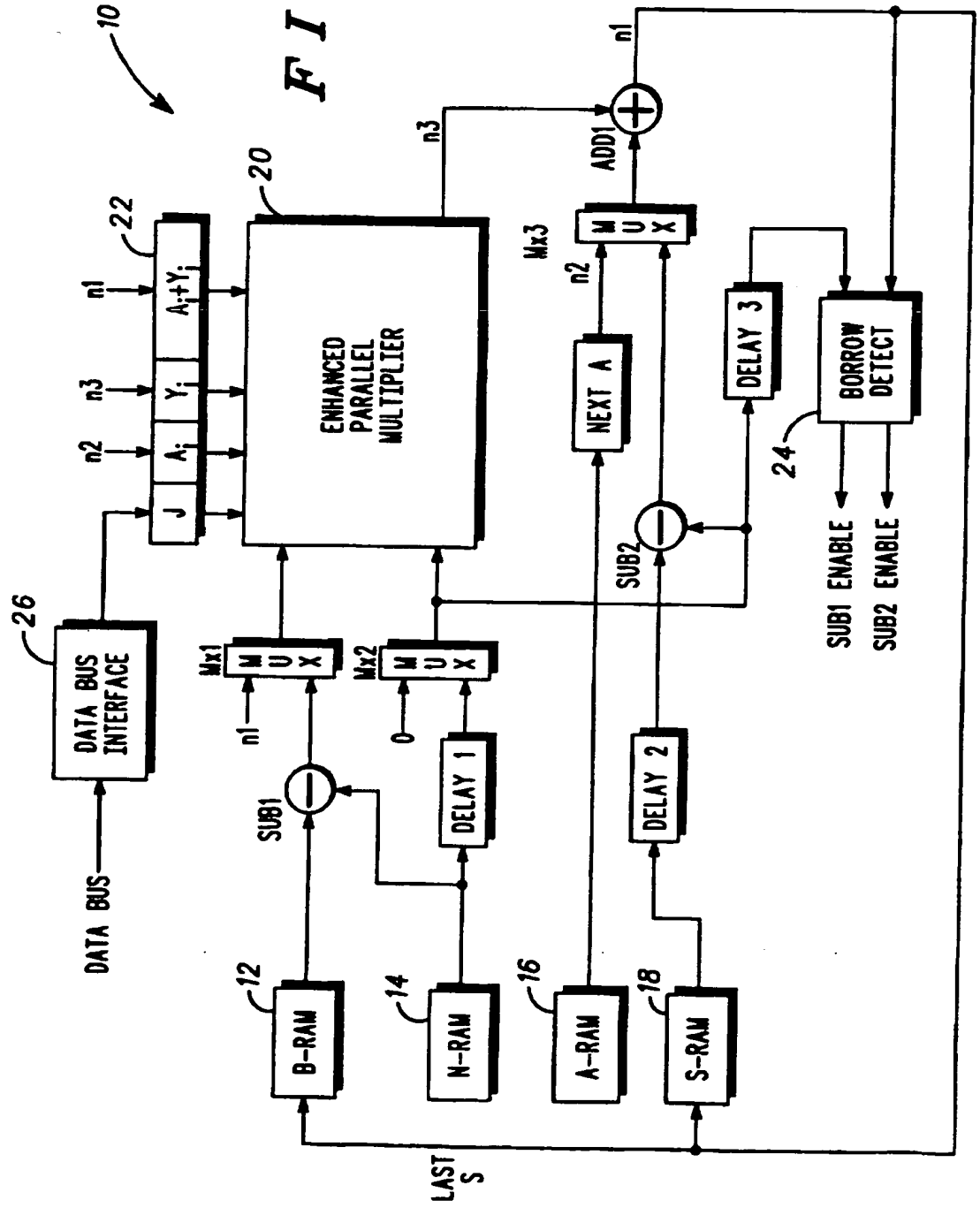


FIG.1
—PRIOR ART—

FIG. 2



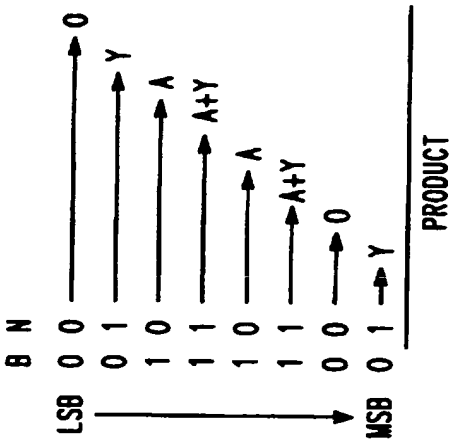


FIG. 4

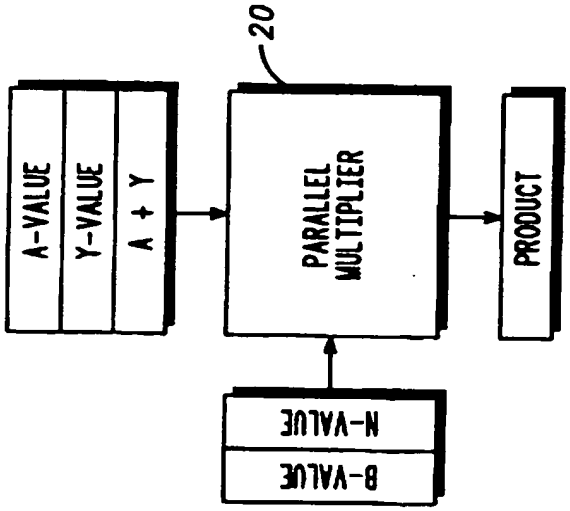


FIG. 3

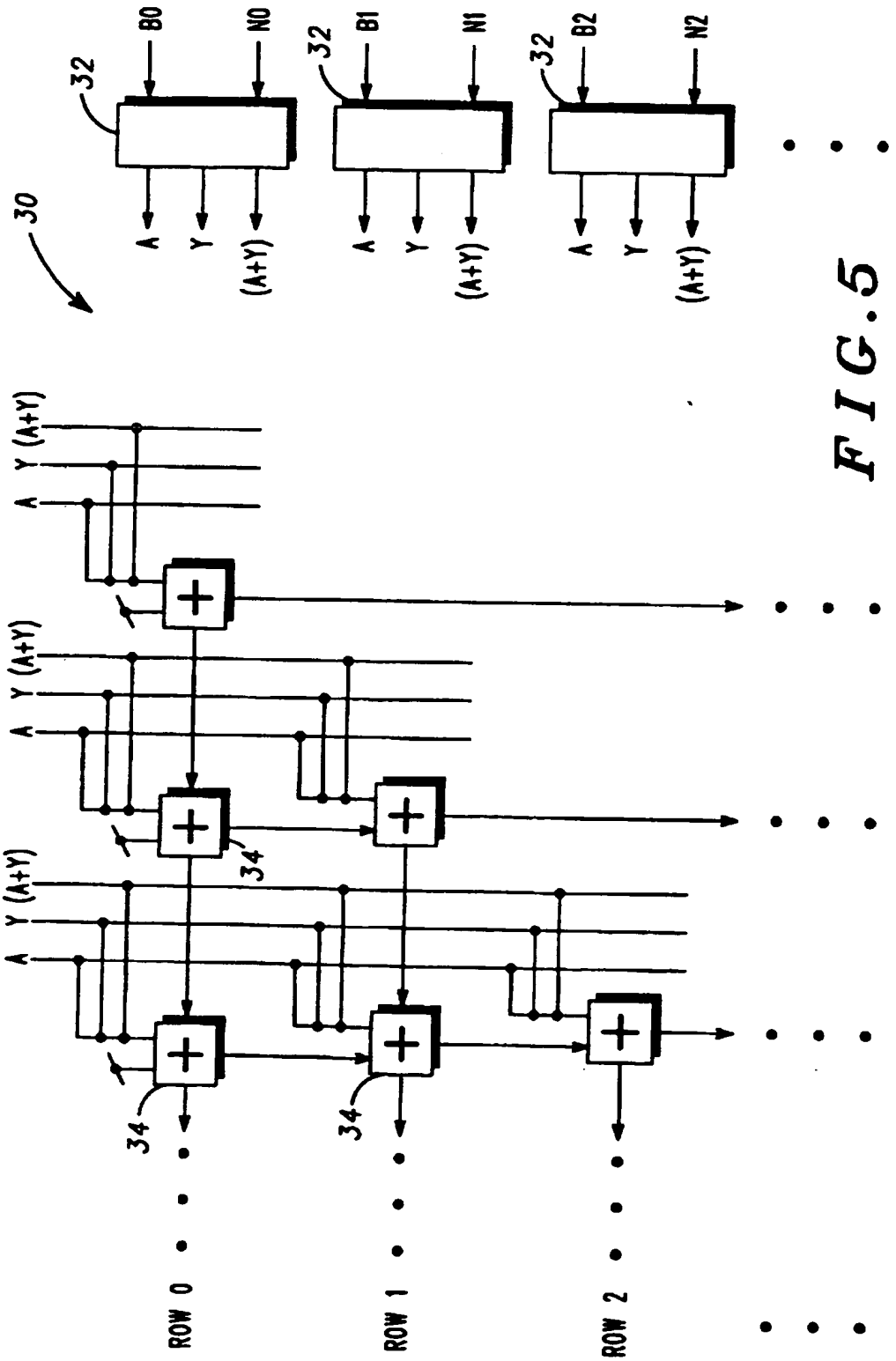


FIG.5